Langara College
CPSC 2150
Assignment #8: Graphs

Assignment due with Brightspace at 11:50pm on April 4

Read chapter §8.1, §8.2, §8.3, § 8.4, §8.5 in Drozdek and in Goodrich §13.1, §13.2, §13.3

Purpose

- To implement graph algorithms using the *adjacency list* or *list of neighbours* graph representation for a simple undirected graph.
- To integrate code written by the student, into an existing program, understanding other people's code.

Implement a class Graph that provides the following functionality:

- creates an empty graph
- reads the edges from an input stream and builds a graph representation
- prints the graph using an output stream
- returns how many vertices are in the graph
- determines if the graph is connected using a breadth first search algorithm
- determines if the graph has at least one cycle
- prints (lists) all the connected components of the graph:
    - a connected component is a subset of the vertices that are connected
    - a singleton vertex by itself is a connected component
    - a graph that is not connected can have connected components
- calculates the path length between two vertices

About your implementation:

- The adjacency list G or list of neighbours for this assignment
    - is a dynamically allocated array of n pointers.
    - the array has n linked lists (which could possibly be empty)
    - each node has a neighbouring vertex and a pointer to the next node:
      so G[v] will be the linked list of neighbours of the vertex v.

  Do NOT use the STL vector[1].
  Do not use variable length arrays as these are not standard ANSI C++.

[1] Yes, yes, an STL vector could have been used but we still need practice in programming memory deallocation, destructors and in preventing dangling pointers

- overload the input operator >> so that the graph can be constructed using data obtained from an input stream
  After the graph has been read and built, note that you are *not* going to support the functionality of adding or removing an edge or a vertex. Before the graph data is read using the operator >>, the previous graph is destroyed if there is a graph already.

- overload the output operator << to output the graph
  the format for the output is not specified

- free up memory that is no longer needed and program the copy constructor, destructor and the overloaded assignment operator

- You MUST do a breadth first search when determining if a graph is connected.

- I do not recommend implementing the cycle detection algorithm as described in Drozdek… think of how else you could do it. There are several ways as discussed in lecture.

- use the Graph.h provided for you and expand the 'private' part of the class: use the documentation of the 'public' part

- do not use the STL with the exception of these (which you may or may not need): std::string, std::stack, std::queue, std::priority_queue

Example of a data input file data1.txt

```
7
0 1
1 2
2 3
2 4
4 3
this graph has a cycle, is NOT connected and has as
components
0 1 2 4 3
5
6
```

In the input file above, the first 7 indicates that there are 7 vertices in the graph: the vertices are labelled 0 1 2 3 4 5 6
so in general for n vertices, the vertices are labelled 0 1 … n-1

0 1 is the undirected edge from vertex 0 to vertex 1 and so on

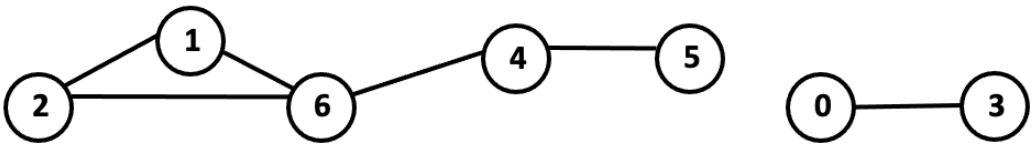
you can read with >> into an int until the stream fails and that's why you can put comments at the bottom of the file that will be ignored. In this case I have put that the graph does have a cycle, is not connected and that the connected components are the set of vertices {0, 1, 2, 4, 3}, the set {5} and the set {6}.

Except for the optional comments at the bottom of the file, use this file format for your program namely, the graph file will have the number of vertices n followed by edges as pairs of vertices labelled 0 1 … n-1

pathLength returns the length of **a** path from the starting vertex vStart in G to the destination vertex vDest in G.
If there is no path from vStart to vDest, pathLength should return -1
If vStart is equal to vDest, pathLength should return 0

Example

Here is a graph with 7 vertices labeled 0, 1, 2, 3, 4, 5, 6 and the undirected edges: edge (2,6), edge (6,4), edge (1,2), edge (1,6), edge (4,5), edge (3,0).



The call to pathLength with vStart=2 and vDest=4, could correctly return 3 or it could correctly return 2

Application File

An application file is provided for you: it is in the file called solver.cpp
It is a simple command driven (menu style) program that tests your Graph class. See the documentation on the application file.

Your code must comply with the interface of solver.cpp given in Graph.h

There are also google unit tests. As in previous assignments, labs and midterms

```
make
```

compiles and links the unittests whereas

```
make test
```

compiles and links the program solver.

A sample commands file can be run[2] from the shell

```
solver -batch < cmds.txt > output.txt
```

(the file output.txt is a reference for you produced from using cmds.txt)

[2] the Windows PowerShell does not like file indirection with <

To submit as Assignment #8 as a single compressed zip file

i. the source code
   a. Graph.h expand big time in the private part of the class
      if you declare a node as a class, submit the necessary files and fix the Makefile provided
   b. Graph.cpp implement the functions
      add the missing code and other functions and variables
   c. myOutput.txt
      which is a result of running
      solver -batch < cmds.txt > myOutput.txt
   d. README.txt if needed
      i. parts not implemented
      ii. known bugs

submit in the same zip file (after having done make clean or make remove)
Makefile
data0.txt data1.txt data2.txt data3.txt data4.txt data5.txt data6.txt data7.txt data8.txt data9.txt
cmds.txt
output.txt
solver.cpp
Commands.cpp
Commands.h
unittest_Graph.cpp